MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
RAČUNARSTVO

ANTONIO KOVAČEVIĆ

BEŽIČNI PREKIDAČ ZA SUSTAV PAMETNE KUĆE UPRAVLJAN PUTEM RAČUNALNE MREŽE TEMELJEN NA MIKROUPRAVLJAČU ESP8266

ZAVRŠNI RAD

ČAKOVEC, 2016

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
RAČUNARSTVO

ANTONIO KOVAČEVIĆ

BEŽIČNI PREKIDAČ ZA SUSTAV PAMETNE KUĆE UPRAVLJAN PUTEM RAČUNALNE MREŽE TEMELJEN NA MIKROUPRAVLJAČU ESP8266

WIRELESS SWITCH AS A PART OF SMART HOME SYSTEM CONTROLLED BY COMPUTER NETWORK IMPLEMENTED WITH ESP8266 MICROCONTROLLER

ZAVRŠNI RAD

Mentor:
dr. sc. Mihael Kukec, v. pred.

ČAKOVEC, 2016

Sažetak

Ovim radom adresiraju se specifičnosti ostvarenja bežičnog prekidača u smislu implementacijskih izazova pri izradi prototipa takvog rješenja te integriranju sklopovskih i programskih rješenja u jedinstven sustav. Rad se bavi izradom prekidača za sustave pametne kuće koji demonstrira mogućnost njegovog upravljanja putem računalne mreže temeljene na protokolu TCP/IP te uz korištenje tzv. MQTT protokola za komunikaciju. Sklopovski dio sustava sastoji se od integriranog kruga ESP8266 i razvojnog sustava „Raspberry Pi" na kojemu je instaliran operacijski sustav Windows IoT Core. Navedene su najvažnije značajke razvojnog sustava Raspberry Pi te integriranog sklopa ESP8266 i njegove usporedbe sa Arduinovim ATmega328P mikroupravljačem koji se koristi na razvojnoj platformi Arduino UNO. Opisuju se koraci implementacije, od aplikacije instalirane na Windows IoT Core operacijskom sustavu sve do krajnjeg modula ESP8266 na kojeg se povezuju uređaji kojima se udaljeno upravlja. U sklopu rada izrađena su dva prototipna rješenja– WiFi prekidač koji dozvoljava upravljanje jednim izlazom, odnosno relejom, preko lokalne bežične mreže i Wifi relej pločica koja, uz dodatne integrirane sklopove, dozvoljava kontrolu nad osam zasebnih izlaza preko lokalne bežične mreže. Također, opisuju se dvije različite metode pristupa problemu kod izrade aplikacije za razvojni sustav Raspberry Pi. Kod izrade prvog prototipnog rješenja koriste se dvije aplikacije za Windows IoT operacijski sustav – web poslužitelj koji poslužuje web aplikaciju korisniku, i samostalna aplikacija koja prebaci stanje izlaza u ovisnosti o naredbama koje dobiva od web aplikacije, dok aplikacija za drugo prototipno rješenje objedinjuje navedene dvije aplikacije u jednu. Opisane su poteškoće koje su nastale tokom izrade prototipnog rješenja te su obrazložene komponente koje su korištene u rješavanju istih. Prototipna rješenja, koja koriste ESP8266 integrirani sklop, su programirani na dva načina – koristeći Arduino razvojnu platformu, odnosno koristeći Sming framework. Opisana je struktura MQTT protokola, princip njegovog rada, način ostvarivanja veze te razmjene podataka između klijenata preko tzv. MQTT brokera. Također, opisuju se razine održavanja kvalitete te vrste zamjenskih znakova koje MQTT komunikacijski protokol podržava.

Ključne riječi: *IoT, pametna-kuća, MQTT, automatizacija, raspberry-pi*

Sadržaj

1. Uvod

Pametne kuće koriste jedno ili više računala kojima kontroliraju osnovne funkcije i značajke kuće, automatski i na daljinu. Sustav pametne kuće može sadržavati regulaciju grijanja i hlađenja u kući, podizanje i spuštanje zastora, upravljanje sustavom za zaštitu kuće te upravljanje rasvjetom. Također, takvim se sustavom može i upravljati s bilo kojeg mjesta uz pretpostavku da sustav podržava upravljanje na daljinu te je povezan na internet [1].

'Pametne' kuće su započele s primjenom početkom 20 stoljeća. Ti sustavi su se većinom koristili u vojnoj industriji te nisu bili javno dostupni. S uvodom protokola X10 1975. godine, prve generacije uređaja koje dozvoljavaju kontrolu nad uređajima su postale dostupne građanima, ali pri visokoj cijeni. Međutim, kako je vrijeme prolazilo tako je sve više domova sadržavalo automatizirane sustave čija se cijena sve više smanjivala [2].

Suvremen trendovi u području su primjena otvorenih sustava te korištenje novih inovativnih tehnologijama koje omogućuju automatizaciju kuće pri prihvatljivoj cijeni. Mnogi se pojedinci sami upuštaju u izradu vlastitih rješenja pametnih kuća čemu je pogodovala prihvatljiva cijena komponenti za izgradnju sustava te široka rasprostranjenost gotovih sklopovskih i programskih komponenti.

2. Sklopovske komponente korištene pri izradi prototipa

Dvije su osnovne sklopovske komponente korištene pri izradi prototipnog rješenja. Prva je razvojni sustav „Raspberry Pi 2“ temeljen na tzv. „sustavu na čipu“ (engl. *System on a Chip* – SOC) tvrtke Broadcom naziva BCM2835. Druga najvažnija komponenta je sklopovski modul za bežičnu komunikaciju prema standardu IEEE 802.11 b/g/n, naziva ESP8266. Obje sklopovske komponente su detaljnije opisane u narednim poglavljima

2.1 Razvojni sustav „Raspberry Pi“

Razvojni sustav Raspberry Pi je kompaktna pločica koja sadrži sve funkcionalnosti kompletnog računala koji se može priključiti na monitor ili TV te koristi standardnu tipkovnicu i miš. On je primarno namijenjen korištenju u školama za učenje osnova računalnih tehnologija i programiranja [3]. Model korišten u izradi prototipnog rješenja je Raspberry Pi 2 Model B koji ima četverojezgreni ARM Cortex-A7 procesor, VideoCore IV grafički procesor, 1 GB radne memorije i MicroSDHC utor. Ostale specifikacije uključuju 4 USB porta, 40 GPIO pinova, HDMI, sučelje za kameru (CSI), sučelje za zaslon (DSI) i 3.5 mm audio izlaz. [3]

Slika 1. Raspberry Pi 2 Model B
Izvor: www.raspberrypi.org

Razvojni sustav Raspberry Pi ne dolazi s instaliranim operacijskim sustavom, nego se on instalira i pokreće s memorijske SD kartice. Raspberry Pi primarno koristi operacijski sustav baziran na Linux-kernelu dok je u novije vrijeme dostupna inačica Windows operacijskog sustava s podrškom za razvoj univerzalnih aplikacija. Operativni sustav službeno podržan od strane Raspberry Pi zaklade je Raspbian, a riječ je o Debian baziranoj Linux distribuciji optimiziranoj upravo za Rapsberry Pi. Neki od poznatijih alternativnih Raspberry Pi operativnih sustava su OpenElec (medijski centar), Pidora (Fedora distribucija), RISC OS (operativni sustav dizajniran specifično za ARM procesore), Android te operativni sustav na kojem se bazira ovaj rad, Windows IOT Core. [4]

2.2 Modul za bežičnu komunikaciju ESP8266

Integrirani sklop (engl. *Integrated circuit - IC*) ESP8266 je druga iznimno važna komponenta koja je korištena prilikom izrade ovdje opisanog prototipa. Može se reći da IC ESP8266 predstavlja veliki napredak u svijetu elektroničkih sklopova vezanih uz razvoj rješenja za tzv. Internet stvari (engl. *Internet of Things – IoT*) . Navedeni sklop utjelovljuje kompletan 32-bitni sustav na čipu s ugrađenom flash memorijom, radnom memorijom i EEPROM-om s podrškom za bežičnu komunikaciju prema IEEE 802.11 b/g/n protokolu. Posebno je važno istaknuti mogućnost njegovog programiranja na dva načina:(i) korištenjem alata za razvoj softvera za sklopovski platformu Arduino, (ii) korištenjem programskog jezika Lua [5].

Slika 2. ESP8266 IC
Izvor: www.olimex.com

Prije pojavljivanja sklopa ESP8266 koristio se sklop Arduino koji je važan doprinos zajednici hardvera otvorenog koda i polazna točka za većinu hobista koji žele ući u svijet 'uradi sam' električnih krugova i procesora. U tablici 1 navedene su karakteristike ESP8266 integriranog sklopa i Arduinovog ATmega328P mikroupravljača te se može vidjeti njihova međusobna razlika. Može se zaključiti da su karakteristike ESP8266 IC-a puno naprednije u odnosu na ATmega328P i da se u određenim primjenama Arduino može zamijeniti s ESP8266 integriranim sklopom zbog svoje manje potrošnje energije, manjeg formata i ugrađenom podrškom za povezivanje na WiFi mrežu. [6]

Tablica 1. Usporedba karakteristika ESP8266 i ATmega328P mikroupravljača [6] [7]

	ESP8266	*Arduino UNO (ATmega328P)*
Radni napon	3.3V	5V
Flash memorija	512KB-4MB	32 KB
SRAM	64 KB	2 KB
ROM	64 KB	4KB
Radni takt	80 MHz	16 MHz
Digitalni I/O pinovi	16	14
Analogni I/O pinovi	1	6
WiFi	802.11 b/g/n	N/A

Uz osnovni komplet za razvoj softvera za ESP8266 (engl. *Software development kit – SDK*) od proizvođača „espressif“, ESP8266 integrirani sklop se može programirati na više načina od kojih su neki opisani u daljnjim odlomcima.

2.2.1 Programiranje integriranog sklopa ESP8266 korištenjem Arduino IDE-a

Nedugo nakon popularizacije integriranog sklopa ESP8266, zahvaljujući ESP8266 društvenoj zajednici, napravljena je podrška za programiranje integriranog sklopa ESP8266 u Arduino razvojnom okruženju. Sve što je potrebno da bi Arduino IDE bio spreman za programiranje ESP modula je instalirati ESP8266 proširenje direktno iz Arduinovog upravljača razvojnih ploča (engl. *Bord Manager*) te odabrati ESP8266 modul na popisu instaliranih ploča.

```
bool value = true;
void blink() {
  digitalWrite(LED_BUILTIN, value);
  value = !value;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  blink();
  delay(1000);
}
```

Slika 3. Programski kôd za paljenje i gašenje LED diode u intervalu od jedne sekunde napisane koristeći Arduino IDE.
Izvor: autor

2.2.2 Programiranje integriranog sklopa ESP8266 korištenjem programskog jezika Lua

Druga metoda programiranja modula ESP8266 je pomoću programskog jezka Lua. Lua je svestran programski jezik koji je dostupan u okruženju za razvoj softvera za sklop ESP8266. Najpopularnija implementacija programskog jezika Lua za ESP8266 je poznata kao mikroprogram „NodeMCU Lua firmware“, koji je dostupan na svojoj GitHub stranici [8]. Taj mikroprogram (engl. *firmware*) se postavlja na modul ESP8266 koristeći bilo koji od alata koji su namijenjeni za prijenos mikroprograma na mikroupravljač. [9]

```
local pin = 4
local value = gpio.LOW
function blink ()
    if value == gpio.LOW then
        value = gpio.HIGH
    else
        value = gpio.LOW
    end
    gpio.write(pin, value)
end
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, value)
tmr.alarm(0, 1000, 1, blink)
```

Slika 4. Programski kôd za paljenje i gašenje LED diode u intervalu od jedne sekunde napisane koristeći Lua programski jezik.
Izvor: autor

2.2.3 Programiranje integriranog sklopa ESP8266 korištenjem razvojnog okvira Sming

Sming je razvojni okvir otvorenog kôda koji je namijenjen za razvoj mikroprograma visoke učinkovitosti za IC ESP8266 koristeći C++ programski jezik. Sming koristi ESP8266 SDK i time, za razliku od Arduino koda za pokretanje programa (engl. *bootloader*), ima mali overhead[1] [10]. Sming je temeljen na događajima (engl. *events*), za razliku od Arduinovog koncepta temeljenog na petlji.

```
Timer procTimer;
bool value = true;
size_t pin = 2
void blink(){
    digitalWrite(pin, value);
    value = !value;
}
void init() {
    pinMode(pin, OUTPUT);
    procTimer.initializeMs(1000, blink).start();
}
```

Slika 5. Programski kôd za paljenje i gašenje LED diode u intervalu od jedne sekunde napisane koristeći Sming framework.
Izvor: autor

[1] Resursi (najčešće memorija i CPU) koji se koriste, ali ne doprinose direktno na željeni rezultat, ali su obavezni.

3. Operacijski sustav „Windows IoT Core“

Microsoft Windows 10 operacijski sustav izrađen je kao više-platformski operacijski sustav, koji će korisnik na sličan način rabiti bez obzira koristi li se neki pametan telefon ili tablet, standardni prijenosnik ili PC računalo ili pak računalo koje nema monitor i tipkovnicu. Jedna od inovacija su Univerzalne aplikacije. Napušta se programsko sučelje Win32 (Win32 API) koje se koristilo dugi niz godina za izradu aplikacija za stolna računala i koristiti se novo „Windows Runtime“ programsko sučelje (WinRT API), početno predstavljeno kao Metro, nakon čega je često mijenjao ime, te je trenutno zadnji naziv koji se koristi „Universal Windows Platform“ ili UWP. Pri izradi prototipnih rješenja opisanih u ovom radu je verzija Windows 10 operacijskog sustava je Windows IoT Core [11].

Windows IoT Core je ograničena verzija Windows 10 operacijskog sustava ciljana za Internet stvari te nije namijenjena za izvođenje aplikacija kakve se uobičajeno nalaze na uredskim ili kućnim računala. Primarna namjena joj je izvođenje na specijaliziranom sklopovlju koje će se u budućnosti ugrađivati u mnoge uređaje i stvari prisutne u domovima kako bi im se omogućila međusobna interakcija i razmjena podataka s ciljem olakšavanja uobičajenih zadataka njihovim korisnicima. Operacijski sustav Windows IoT Core ne podržava klasične aplikacije temeljene na programskom sučelju Win32 (primjerice Microsoft Office, Chrome i Notepad se neće moći instalirati i pokrenuti). Nova instalacija operacijskog sustava Windows 10 na Raspberry Pi-u ne pokreće se u svima poznatom desktop okruženju, nego će Windows IoT Core pokazati korisnicima samo jednu univerzalnu Windows aplikaciju. Operacijski sustav može prikazivati sučelje samo jedne aplikacije istovremeno, premda se dodatni procesi mogu izvršavati u pozadini. Programski jezik koji se koristi za stvaranje aplikacija je C# a razvojna okolina je Microsoft Visual Studio 2015. Uz programski jezik C# aplikacije mogu biti pisane koristeći razne jezike kao što su Visual Basic sa XAMLom, JavaScript sa HTMLom, ili C++ sa DirectXom i/ili XAMLom. [12]

4. Komunikacijski protokol MQTT

Protokol MQTT (engl. *Message Queue Telemetry Transport*) protokol baziran na principu objavi-pretplati (engl. *publish/subscribe*). Model objavi-pretplati je alternativa tradicionalnom modelu klijent-poslužitelj, gdje klijent komunicira izravno s krajnjim točkama. Međutim, model objavi-pretplati odvaja klijenta, koji šalje određenu poruku, zvan izdavač (engl. *publisher*) od drugog klijenta koji prima poruku, zvan pretplatnik (engl. *subscriber*). U tom modelu pretplatnici ne znaju jedan za drugog. Tu dolazi i treća komponenta, zvana broker, koja je poznata i od strane izdavača i pretplatnika, koji filtrira sve dolazne poruke i distribuira ih u skladu s tim. [13].

Protokol MQTT je vrlo jednostavan protokol za razmjenu poruka, oblikovan za uređaje niskih performansi i niske propusnosti na nepouzdanim mrežama ili mrežama visoke latencije. Prilagođen je radu u uvjetima smanjene propusnosti mreže i ograničenih resursa uređaja te istovremeno pokušava osigurati pouzdanost i stupanj osiguranja isporuke poruke. Te karakteristike čine protokol MQTT idealnim za komunikaciju između uređaja (engl. *Machine-to-Machine – M2M*) ili u svijetu interneta stvari (IoT) te za mobilne uređaje gdje su propusnost mreže i snaga baterije resursi koje je potrebno optimalno koristiti. [14]

4.1 Razlika između MQTT i HTTP komunikacijskog protokola

Iako su česte usporedbe između MQTT i drugih sličnih protokola, najkorisnija usporedba je s HTTP (čiji se detalji mogu vidjeti u tablici 2) zbog velike raširenosti HTTP protokola kod prijenosa web sadržaja te razlike između modela razmijene poruka na kojima se baziraju spomenuti protokoli.

Tablica 2. Razlika između MQTT i HTTP protokola [15]

	MQTT	***HTTP***
Dizajn	Prenosi podatke kao slijed bajtova, neovisno o sadržaju.	Ima podršku za MIME standard za definiranje tipa sadržaja.
Model razmijene poruka	Koristi model objavi/pretplati.	Koristi model zahtjev/odgovor .
Kompleksnost protokola	Jednostavna specifikacija. Koristi nekolicinu poruka i metoda. Prikladno za slanje poruka jednostavnog sadržaja.	Sadrži mnogo poruka, metoda i statusnih kodova koji su prikladni za distribuciju multimedijskog sadržaja kao što su audio, video, grafički elementi, tekst i hiperveze.
Veličina poruke	Oblikovan za uređaje s niskom propusnošću. Zaglavlje poruke je samo 2 bajta.	Koristi tekstualni format koji je lakše čitati ljudima i time ga čini jednostavnijim za pronalaženje i rješavanje problema.
Održavanje kvalitete usluga	Podrška za 3 razine održavanja kvalitete usluga.	Nema podršku za provjeru dospijeća poruke.
Distribucija podataka	Ugrađeni mehanizam distribucije podataka. Podrška za 1 do 0, 1 do 1 i 1 do N distribucijskih modela.	Radi na principu od točke do točke. Nema ugrađeni mehanizam za distribuciju podataka.

4.2 MQTT teme

Tema u MQTT komunikacijskom protokolu je UTF-8 kodiran niz znakova koju koristi MQTT broker kako bi filtrirao poruke za svaki od povezanih klijenata. Tema se sastoji od jedne ili više razina te je svaka razina odvojena kosom crtom (/). Klijent nije obavezan stvoriti željenu temu prije objavljivanja i pretplate na nju jer broker prihvaća svaku valjanu temu bez prethodne inicijalizacije. Svaka tema da bi bila valjana mora imati barem jedan znak te može sadržavati razmake. Također, naziv teme je osjetljiv na velika i mala slova, tako da su ***kuća/temperatura*** i ***kuća/Temperatura*** dvije pojedinačne teme. [16]

4.2.1 Zamjenski znakovi

Kada se klijent pretplati na temu, on može koristiti točan naziv teme gdje je poruka objavljena ili se može pretplatiti na više tema odjednom pomoću zamjenskih znakova. Zamjenski znak

može se koristiti samo za pretplate na teme i nije dopušteno prilikom objavljivanja poruke. U nastavku se mogu vidjeti dvije različite vrste zamjenskih znakova.

4.2.1.1 Jednorazinski zamjenski znak

Kao što samo ime već sugerira, jednorazinski zamjenski znak je zamjena za jednu razinu teme. Simbol plus (+) predstavlja jednorazinski zamjenski znak u temi [16]. Na primjer, pretplata na ***kuća/prizemlje/+/temperatura*** će odgovarati ili ne odgovarati sljedećim temama:

- ✓ kuća/prizemlje/dnevniboravak/temperatura
- ✓ kuća/prizemlje/kuhinja/temperatura
- ⊗ kuća/prizemlje/kuhinja/svijetlo
- ⊗ kuća/prvikat/kuhinja/temperatura
- ⊗ kuća/prizemlje/kuhinja/frižider/temperatura

4.2.1.2 Višerazinski zamjenski znak

Dok jednorazinski zamjenski znak pokriva samo jednu razinu teme, višerazinski zamjenski znak (#) pokriva neodređeni broj razina te se on mora uvijek nalaziti na kraju niza teme. Klijent koji se pretplati na temu s višerazinskim zamjenskim znakom prima sve poruke koje počinju s uzorkom prije zamjenskog znaka, bez obzira koliko duboko teme idu [16]. Na primjer, pretplata na ***kuća/prizemlje/#*** će odgovarati ili ne odgovarati sljedećim temama:

- ✓ kuća/prizemlje/dnevniboravak/temperatura
- ✓ kuća/prizemlje/kuhinja/temperatura
- ✓ kuća/prizemlje/kuhinja/svijetlo
- ✓ kuća/prizemlje/kuhinja/frižider/temperatura
- ⊗ kuća/prvikat/kuhinja/temperatura

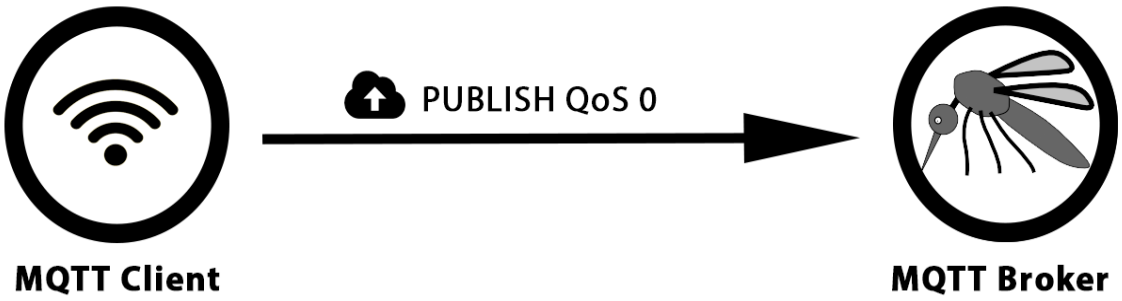
4.3 Razine održavanje kvalitete usluga (QoS)

Razina kvalitete usluge (engl. *Quality of Service - QoS*) je sporazum između izdavača i primatelja poruke u vezi s garancijom dostave poruke. Protokol MQTT podržava tri razine QoS-a:

- Najviše jednom (0)
- Najmanje jednom (1)
- Točno jednom (2)

4.3.1 Razina 0 – najviše jednom

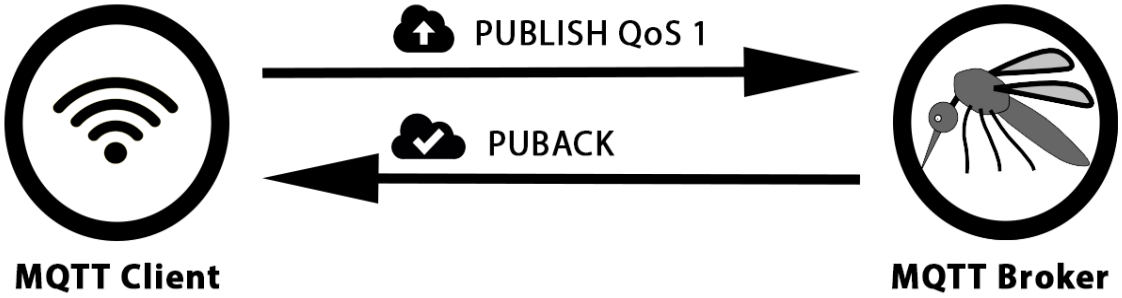
Minimalna razina kvalitete usluge je nula. To osigurava da će poruka biti poslana samo jednom, bez povratne informacije dali je poruka stigla niti ponovnog pokušaja slanja poruke. Poruka stiže primatelju jednom ili nikad. [16]



Slika 6. Komunikacija koristeći QoS 0
Izvor: vlastita izrada od elemenata dostupnih na http://www.iconarchive.com/

4.3.2 Razina 1 – najmanje jednom

Ova kvaliteta usluge osigurava da poruka stigne primatelju barem jednom. QoS 1 PUBLISH paket sadrži paketni identifikator u zaglavlju koji je priznat od strane PUBACK paketa. Kada primatelj zaprimi paket, on pošalje PUBACK paket koji potvrđuje primitak paketa. U slučaju da PUBACK paket ne stigne izdavaču, on će ponovno poslati PUBLISH paket. U tom slučaju može se dogoditi da primatelj zaprimiti paket istog sadržaja. Poruka stiže primatelju jednom ili više puta. [16]



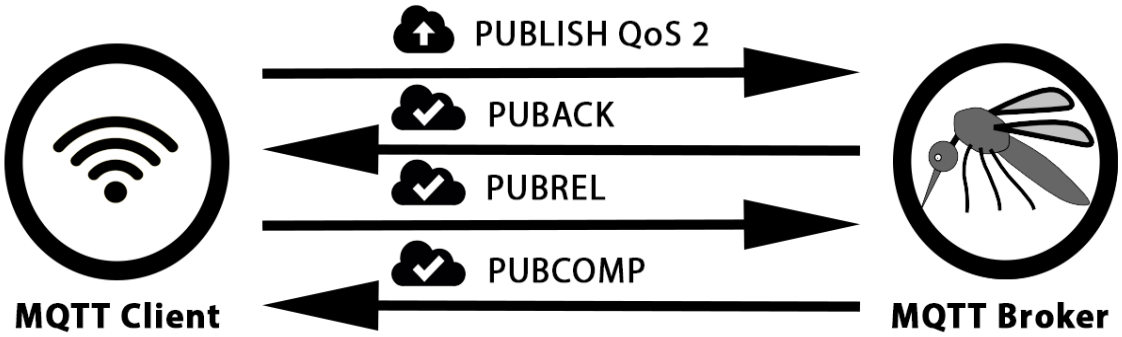
Slika 7. Komunikacija koristeći QoS 1
Izvor: vlastita izrada od elemenata dostupnih na http://www.iconarchive.com/

4.3.3 Razina 2 – točno jednom

Najviša razina kvalitete usluge (QoS 2) jamči da je svaka poruka zaprimljena samo jednom. Takav način komunikacije je najsigurniji ali je ujedno i najsporiji.. Primatelj QoS 2 paketa potvrđuje primitak s procesom priznavanja u dva koraka, što osigurava da će poruka stići jednom i samo jednom.

Primatelj QoS 2 paketa će obraditi primljenu poruku i priznati ga izdavaču s porukom PUBREC. Primatelj će pohraniti referencu na identifikator paketa sve dok ne pošalje paket PUBCOMP. To je važno za izbjegavanje obrade iste poruke po drugi put. Kada izdavač primi poruku PUBREC može sigurno odbaciti objavljenu poruku, jer zna da je primatelj uspješno primio poruku. Sljedeće će pohraniti poruku PUBREC i odgovoriti s porukom PUBREL. Nakon što primatelj dobiva poruku PUBREL, može odbaciti pohranjeno stanje i odgovoriti s porukom PUBCOMP. Isto vrijedi i kada izdavač prima poruku tipa PUBCOMP. Kada proces završi obje strane mogu biti sigurne da je poruka isporučena.

U slučaju da se paket izgubi, izdavač je odgovoran za ponovno slanje zadnje poruke nakon razumnog vremena. To vrijedi kada je izdavač MQTT klijent i također kada MQTT broker šalje poruku. [16]



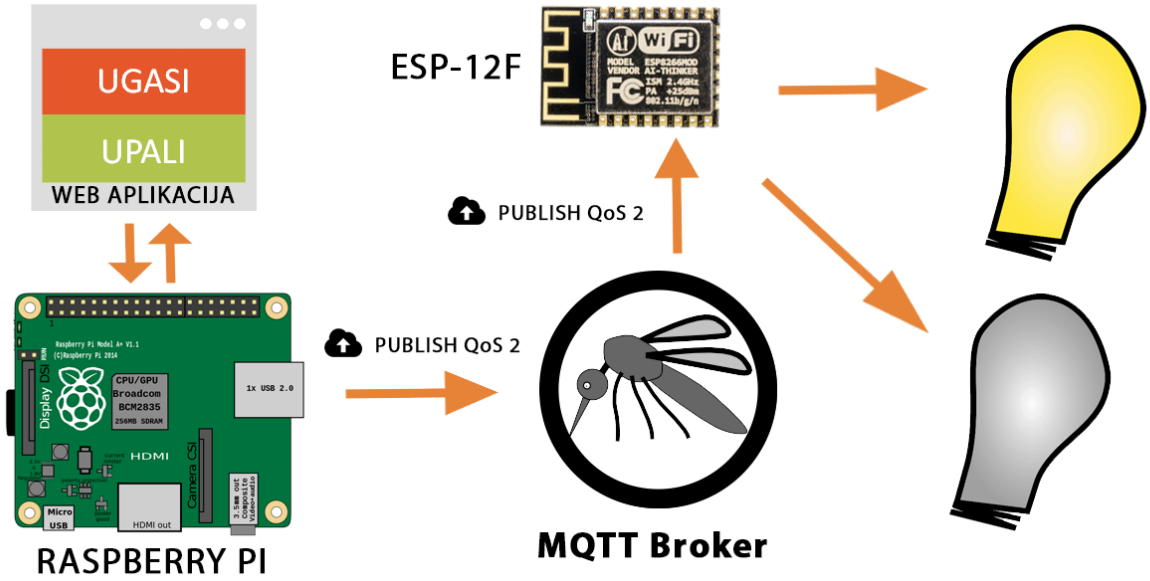
Slika 8. Komunikacija koristeći QoS 2
Izvor: vlastita izrada od elemenata dostupnih na http://www.iconarchive.com/

Ovi prethodni odlomci uvod su u komunikacijski protokol MQTT. Postoje još nekoliko značajki komunikacijskog protokola MQTT koji se ovdje ne spominju, ali su jednako važni (trajne sesije, zadržane poruke, oporuke). Također, ne spominje se sigurnost MQTT protokola (autorizacija pomoću korisničkog imena i lozinke, TLS/SSL, autorizacija X509 certifikatom, OAuth 2.0, enkripcija sadržaja poruke itd.).

5. Izrada prototipnog rješenja - WiFi prekidač

Prije početka same izrade bilo je potrebno osmisliti tok informacija u sustavu koristeći Windows IoT operacijski sustav instaliran na Raspberry Pi-u i ESP-12F WiFi modul.

Na web aplikaciji koju poslužuje web poslužitelj koji se nalazi na Raspberry Pi-u, odabire se željena funkcija, primjerice, želi li se upaliti ili ugasiti žarulju. Nakon odabira, šalje se poruka u predodređenu temu preko bežične mreže (IEEE 802.11) koristeći MQTT komunikacijski protokol. Modul ESP-12F koji je pretplaćen na istu temu, prima poruku od MQTT brokera koji, ovisno o poruci, gasi ili pali žarulju.



Slika 9. Ilustracija toka informacija u sustavu
Izvor: vlastita izrada od elemenata dostupnih na www.raspberrypi.org, www.electrodragon.com, http://www.iconarchive.com/

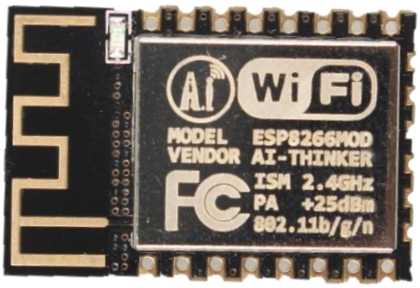
Izrada prvog prototipa je podijeljena u dva djela – dizajn i programiranje krajnjeg uređaja, u ovom slučaju ESP-12F zajedno s relejom i programiranje središnjeg sustava koji upravlja krajnjim uređajima.

5.1 WiFi prekidač

Cilj WiFi prekidača je omogućiti korisniku upravljanje električnim uređajem, bila to rasvjeta, kuhalo za vodu ili pegla, preko bežične mreže. To se ostvarilo pomoću releja i mikroupravljača s ugrađenom podrškom za bežične mreže prema standardu IEEE 802.11. Relej korišten u izradi prototipnog rješenja je elektromagnetni monostabilni SPDT (engl. *Single pole, double throw*) relej naziva SRD-03VDC-SL-C od proizvođača Songle Relay. Korišteni relej je monostabilni što znači da se zavojnica vraća u prvobitni položaj nakon prestanka dotoka električne struje.

5.1.1 Dizajn

Dvije najvažnije komponente WiFi prekidača su modul ESP-12F i relej (SRD-03VDC-SL-C). Modul ESP-12F je proizvod tvrtke AI-Thinkers koji sadrži ESP8266 mikroupravljač. Sam dizajn WiFi prekidača je relativno jednostavan i može se vidjeti na slici 12, ali kod oblikovanja bežičnog prekidača moralo se pripaziti na nekoliko stvari:



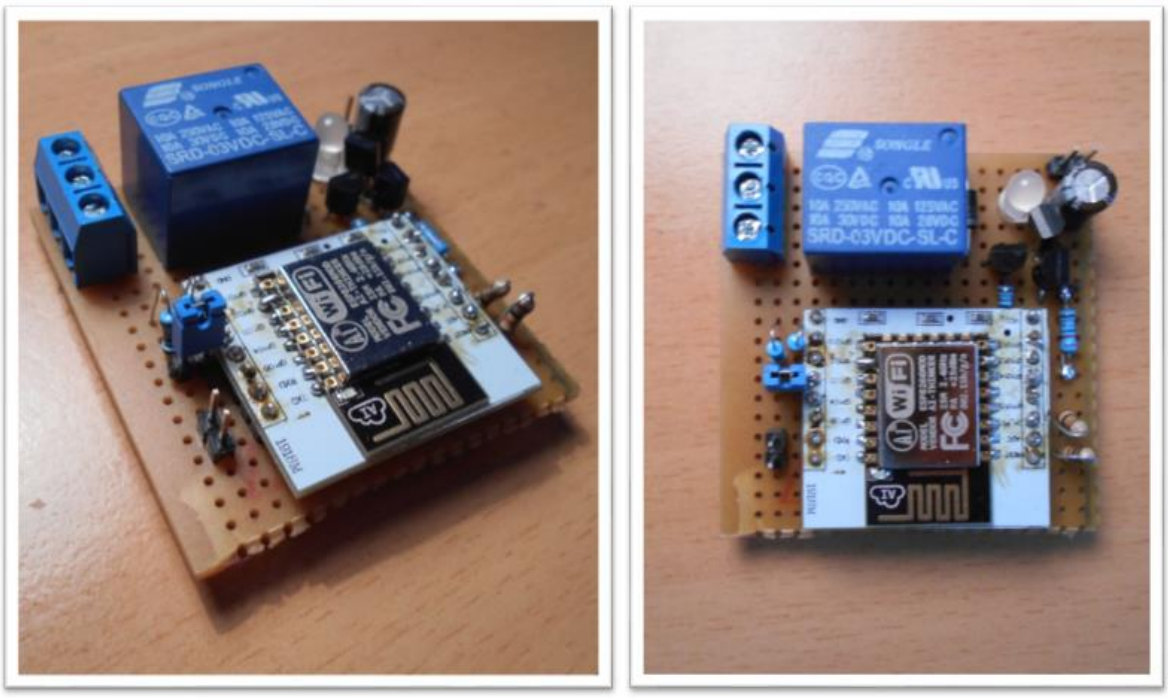
Slika 10. ESP-12F modul
Izvor: www.electrodragon.com

1. Magnetnoj zavojnici u releju je potrebno minimalno 120mA da bi se aktivirao kontakt [17], međutim ESP8266 može pružiti samo 12mA struje [6]. Rješenje tog problema je korištenje tranzistora, u ovom slučaju P2N2222A, kao prekidača. Time mikroupravljač ne upravlja relejom direktno, nego služi samo kao okidač.
2. Zbog činjenice da relej koristi zavojnicu, može doći do naglog porasta napona obrnutog polariteta. Kada je zavojnica pod naponom, ona stvara magnetno polje. Kada se prekine izvor napona, magnetsko polje se uruši i stvara napon obrnutog polariteta. To stvara prolazni naponski impuls koji može imati vrijednosti mnogo puta veću od izvornog napona te može oštetiti druge komponente u strujnom krugu koje nisu napravljene da se mogu nositi s povratnim naponom određene vrijednosti.

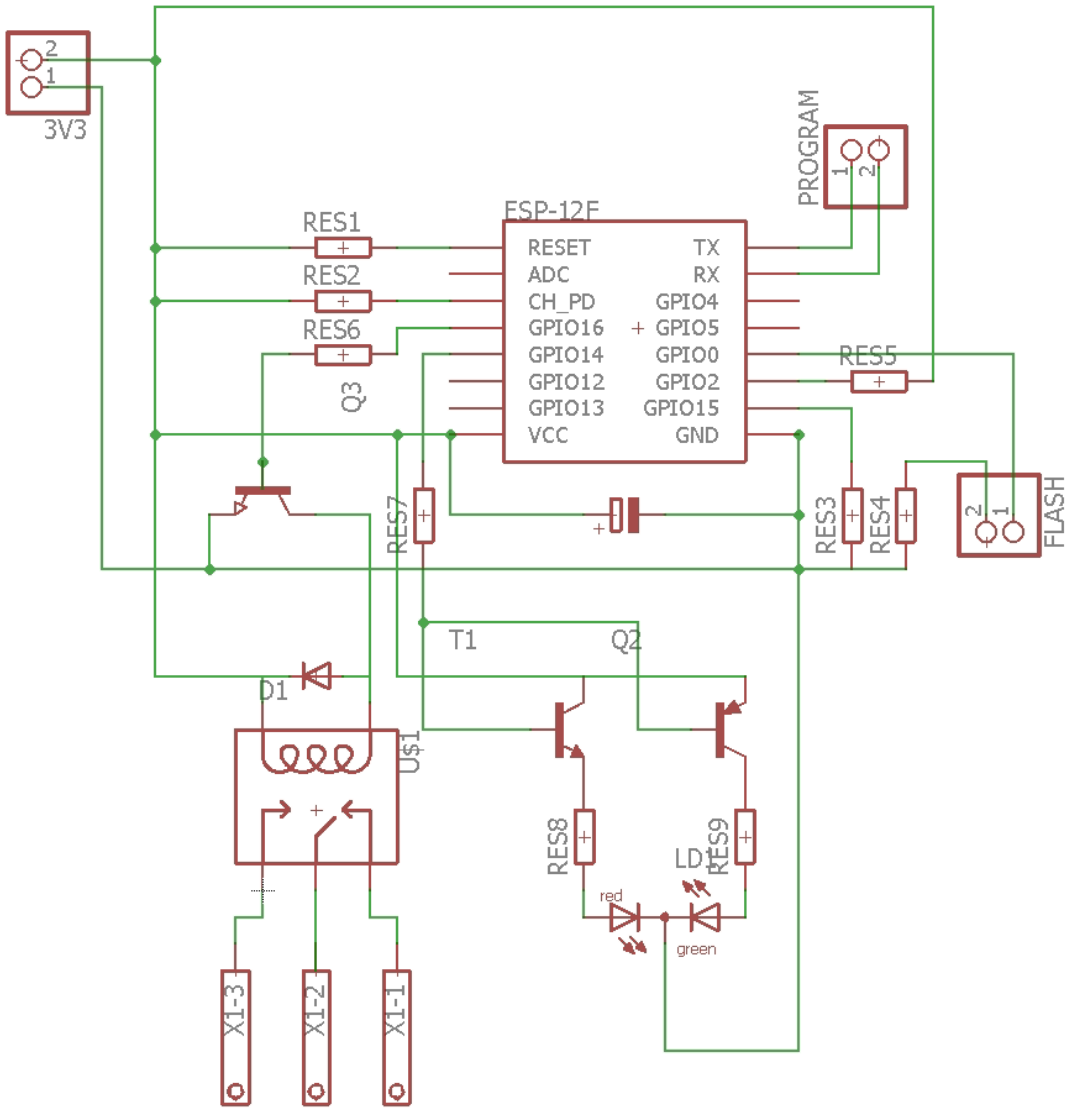
Stavljanje obrnuto-pristrane diode preko zavojnice dozvoljava naponskom impulsu da se sigurno raspršti u otporu svitka zavojnice [18].

3. Kod paljenje/gašenja zavojnice, LED diode, mikroupravljača, bilo kakva nagla promjena opterećenja i općenito djelovanje elektroničnih elemenata stvara buku – fluktuaciju u električnom signalu. Svaki od elementa ima određenu toleranciju za fluktuacije, od nisko do visoko osjetljive. Kako bi se smanjile fluktuacije stavlja se kondenzator preko VCC i GND porta modula ESP-12F, jer je on najosjetljiviji od ostalih elemenata.

Wifi prekidač sadrži i DUO-LED diodu koja u sebi ima dvije LED diode sa zajedničkom katodom i time može emitirati svijetlost u dvije različite boje, u ovom slučaju žutu i zelenu. Uz to, WiFi prekidač ima tri para pinova, jedan služi da spajanje izvora napajanja (3.3V/GND), drugi za spajanje RC i TX linije za prijenos mikroprograma na mikroupravljač i zadnji za omogućavanje prijenosa.



Slika 11. Izgled WiFi prekidača.
Izvor: autor



Slika 12. Shema Wifi prekidača izrađena u softverskom paketu Eagle
Izvor: autor

5.1.2 Mikroprogram

Odabrana metoda za programiranje ESP-12F modula je korištenje razvojne platforme Arduino koja je odabrana zbog svoje jednostavnosti, kompatibilnosti s odabranim knjižnicama programskog kôda za Arduino te je poznatog IDE okruženja. Računalni program u razvojnom okruženju Arduino naziva se „skica“ (engl. *sketch*) pa će se taj naziv koristiti i ovdje u tekstu. Na vrhu skice se uključuju biblioteke koje će se koristiti u skici, u ovom slučaju su to ESP8266 WiFi, jednostavna ESP8266 Arduino biblioteka i biblioteka programskog koda naziva „PubSubClient“ koja omogućava jednostavnu komunikaciju sa serverom koji podržava MQTT protokol.

Skica sadrži par ugrađenih funkcija. Jedna od tih je funkcija naziva „setup“, koja se poziva kad se skica pokrene. Koristi se da bi se inicijalizirale varijable, postavio način rada pinova itd. Funkcija „setup" se poziva samo jednom kod svakog uključivanja ili resetiranja mikroupravljača. U toj funkciji se postavlja brzina komunikacije preko serijskog izlaza ESP8266 IC-a pomoću funkcije naziva „Serial.begin()“, postavljaju se podaci na koji MQTT server će se modul povezati.

Funkcija „loop“ (u prijevodu petlja) je druga funkcija ugrađena u Arduino skicu, sukladno imenu, radi se o beskonačnoj petlji u kojoj se izvodi cijeli program. Unutar te funkcije nalazi se uvjet koji u slučaju da modul nije ostvario vezu s MQTT serverom, poziva funkciju naziva „reconnect“ kojom se uspostavlja veze sa serverom.

```
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");
    for (int i = 0; i < length; i++){
        Serial.print((char)payload[i]);
    }
    Serial.println();

    if ((char)payload[0] == '1') {
        FlipLED(1);
    } else {
        FlipLED(0);
    }
}
```

```
void reconnect() {
    while (!client.connected()){
        setColor('O');
        Serial.print("Attempting MQTT connection...");
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            setColor('G');
            client.subscribe("mqtt/test2");
        } else {
            Serial.print("failed, rc=" + client.state());
            Serial.println("try again in 5 seconds");
            delay(5000);
        }
    }
}
```

Slika 13. Programski kôd funkcija callback i reconnect.
Izvor:autor

5.1.3 Trošak komponenti

Tablica 3. Ukupni trošak komponenti za izradu bežičnog prekidača (Cijene na dan 23.08.2016)

VETRONIT bušena pločica	33,00 kn
ESP-12F	11,23 kn
SRD-3VDC-SL-C	5,46 kn
Utikač za pločicu plastični 3-POL	1,05 kn
Tranzistori (3x)	1,80 kn
DUO-LED	2,30 kn
Otpornici (5x)	3,20 kn
UKUPNO	**58,04 kn**

5.2 Aplikacija za Windows IoT Core

Za rad sustava potrebne su dvije aplikacije. Web poslužitelj koji će posluživati web aplikaciju u svrhu kontroliranja LED diode kojom se simulira osnovna kontrola nad uređajima te aplikacija koje će prikazivati korisničko sučelje pomoću razvojnog sustava Raspberry Pi-a te prikazivati stanje LED diode. Sama aplikacija je jednostavna. Ona samo prikazuje trenutno stanje LED diode, odnosno je li LED dioda upaljena ili ugašena.

5.2.1 Web poslužitelj

Web poslužitelj poslužuje web aplikaciju pomoću koje se može upravljati izlazom WiFi prekidača. Najvažnija metoda u aplikaciji je naziva „WriteResponseAsync“ koja se poziva nakon što je odabrana željena funkcija, primjerice, želi li se upaliti ili ugasiti žarulju. Glavna funkcija joj je poslati promjenu stanja izlaza aplikaciji „Blinky“.

```
private async Task WriteResponseAsync(string request, IOutputStre
am os)
{
    string state = "Unspecified";
    bool stateChanged = false;
    if (request.Contains("blinky.html?state=on"))
    {
        state = "On";
        stateChanged = true;
    }
    else if (request.Contains("blinky.html?state=off"))
    {
        state = "Off";
        stateChanged = true;
    }

    if (stateChanged)
    {
        var updateMessage = new ValueSet();
        updateMessage.Add("State", state);
        var responseStatus = await appServiceConnection.SendMessa
geAsync(updateMessage);
    }
    …
}
```

Slika 14. Kôd metode „WriteResponseAsync“.
Izvor:autor

5.2.2 Aplikacija „Blinky“

Svrha aplikacije naziva „Blinky“ je poslati poruku koristeći protokol MQTT na predodređenu temu u ovisnosti na poruku primljenu od strane web poslužitelja. Za to su zadužene dvije metode naziva „OnMessageReceived“ i „FlipState“. Metoda „OnMessageReceived“ prima poruku od web poslužitelja te u ovisnosti o poruci, poziva metodu „FlipState“ i mijenja stanje izlaza WiFi prekidača. Aplikacija „Blinky“ koristi biblioteku naziva „M2Mqtt“ – implementacija protokola MQTT za platformu WinRT [19].

```
private void FlipState()
{
    if (Status == 0)
    {
        Status = 1;
        this.client.Publish("mqtt/test2", Encoding.UTF8.GetBytes
("1"));
        LED.Fill = redBrush;
        StateText.Text = "On";
    }
    else
    {
        Status = 0;
        this.client.Publish("mqtt/test2", Encoding.UTF8.GetBytes
("0"));
        LED.Fill = grayBrush;
        StateText.Text = "Off";
    }
}
```

Slika 15. Kôd metode „FlipState“.
Izvor: autor

```
private async void OnMessageReceived(AppServiceConnection sender
, AppServiceRequestReceivedEventArgs args)
{
    var message = args.Request.Message;
    string newState = message["State"] as string;
    switch (newState)
    {
        case "On":
            {
                await Dispatcher.RunAsync(
                CoreDispatcherPriority.High,
                () =>
                {
                    if (Status == 1) { FlipState(); }
                });
                break;
            }
        case "Off":
            {
                await Dispatcher.RunAsync(
                CoreDispatcherPriority.High,
                () =>
                {
                    if (Status == 0) { FlipState();}
                });
                break;
            }
        case "Unspecified":
        default:
            {
                break;
            }
    }
}
```

Slika 16. Kôd metode „OnMessageReceived“.
Izvor: autor

6. Izrada prototipnog rješenja - bežična relej pločica

Drugi prototip prati princip prvoga. Glavna razlika između prvog i drugog prototipa je u broju izlaza s kojim se može upravljati. Kao i kod prvog prototipa, izrada je podijeljena u dva djela – oblikovanje i programiranje krajnjeg uređaja te programiranje središnjeg sustava.

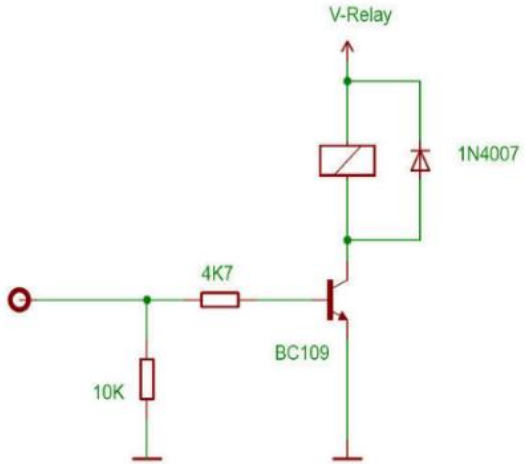
6.1 Bežična relej pločica

Cilj bežične (IEEE 802.11 – WiFi) relej pločice je omogućiti korisniku upravljanje većim brojem električnim uređajem preko bežične mreže. To se ostvarilo pomoću 8 releja (SRD-03VDC-SL-C) i modula s mikroupravljačem s ugrađenom podrškom za bežične mreže pod nazivom ESP-12F.

6.1.1 Dizajn

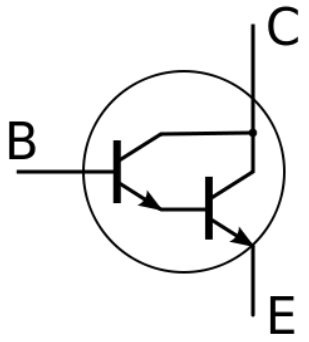
Izrada printane pločice (engl. *Printed circuit bord – PCB*) započela je s dvije osnovne komponente - ESP-12F mikroupravljač i relejom SRD-03VDC-SL-C.

Na slici 15 se može vidjeti što je sve potrebno kako bi se 'upogonio' jedan relej. Uz relej potrebna su dva otpornika, jedan tranzistor i dioda. Pošto WiFi relej pločica koristi 8 releja, broj potrebnih komponenata se uvećava za faktor 8, na ukupno 32 različite komponente. Da bi se smanjio broj komponenata i potreban prostor na PCB-u, koristi se ULN2803ADW – niz od 8 Darlington tranzistora u jednom integriranom krugu.



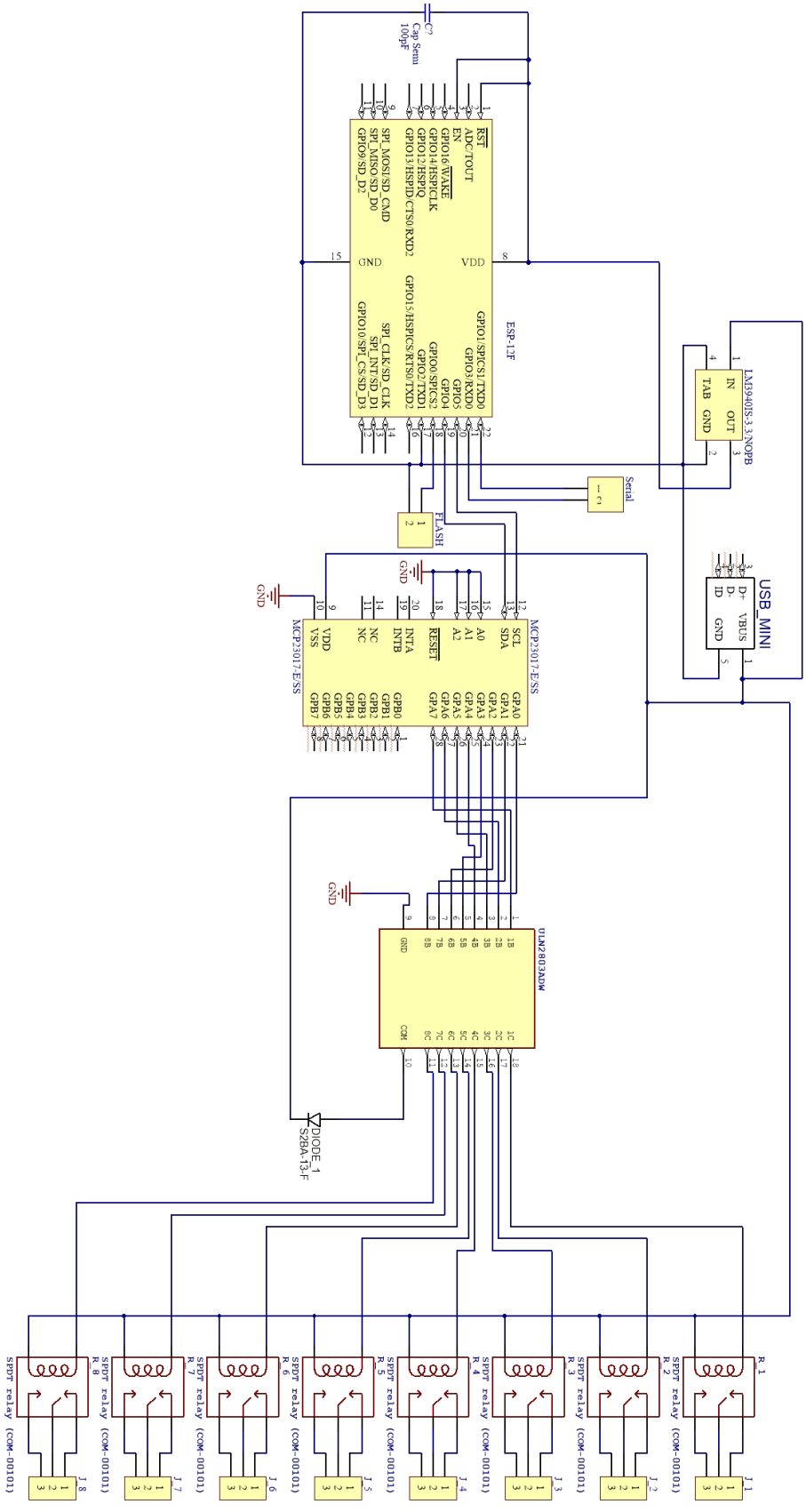
Slika 17. Upravljanje jednim relejom izrađena u softverskom paketu Eagle.
Izvor: autor

Darlington tranzistor, još zvan Darlington par, je konstrukcija koja se sastoji od dva bipolarna tranzistora koji su povezani na takav način da struja koja je pojačana s prvim tranzistorom se dodatno pojačava drugim. Ovakva konfiguracija može dati mnogo veću struju nego svaki tranzistor zasebno.



Slika 18. Darlington tranzistor
Izvor: wikimedia.org

Korištenjem integriranog kruga ULN2803 uvelike je smanjen potreban broj komponenta s 32 na samo jedan, ali postoji još jedan izazov. Modul ESP-12F ima samo 8 iskoristivih ulazno/izlaznih nožica. Pošto su neke ulazno/izlazne nožice potrebne drugdje u dizajnu, ostaje nam nedovoljan broj izlaza za kontrolu releja. Postoji mnogo načina kako proširiti broj GPIO pinova mikroupravljača ali za ovaj projekt koristit će se MCP23017-E/SS IC, koji ima 16 dvosmjernih GPIO pinova te koristi protokol I2C za komunikaciju s mikroupravljačem.



Slika 19. Osnovni dizajn WiFi relej pločice izrađena u softverskom paketu Altium Designer.

Izvor: autor

6.1.2 Mikroprogram

Kod izrade prvog prototipa vidio se nedostatak korištenja Arduino razvojnog okruženja, jedan od tih je ne mogućnost korištenja standardnih C++ biblioteka. Prelazak na Sming razvojno okruženje to omogućuje i kako on koristi SDK tvrtke „espressif“, rasipanje resursa je smanjeno naspram korištenja razvojnog okruženja za Arduino.

6.1.2.1 Spajanje na WiFi mrežu

Ugrađene funkcije u razvojno okruženje Sming pomažu modulu ESP-12F pri spajanju na bežičnu mrežu. Funkcija naziva „WifiStation.config“ postavlja korisničke podatke za spajanje na bežičnu mrežu, dok funkcije naziva „WifiStation.enable“ i „WifiAccessPoint.enable“ postavljaju način operacije integriranog kruga ESP8266. Funkcija naziva „conected“ se poziva kod uspješnog spajanja na mrežu dok kod neuspjeha se poziva funkcija naziva „notConnected“ te se za 20 sekundi ponavlja pokušaj spajanja na bežičnu mrežu.

```
WifiStation.config(WIFI_SSID, WIFI_PWD);
WifiStation.enable(true);
WifiAccessPoint.enable(false);
WifiStation.waitConnection(connected, 20, notConnected);
```

Slika 20. Programski kôd za spajanje na WiFi mrežu.
Izvor: autor

6.1.2.2 Komunikacija koristeći MQTT protokol

Razvojni okvir Sming sadrži ugrađenu podršku za MQTT protokol. Kod uspješnog spajanja na wifi mrežu, MQTT klijent se konfigurira i spaja na MQTT broker te se pretplaćuje na temu „wifi/relayBoard/“. Kod primitka poruke od strane MQTT brokera poziva se funkcija naziva „onMessageReceived“ u kojoj se, ovisno o sadržaju poruke, pali ili gasi jedan od izlaza.

```
byte state = 0b00000000;
MqttClient mqtt("192.168.1.11", 1883, onMessageReceived);
mqtt.connect("esp8266");
mqtt.subscribe("wifi/relayBoard/#");

void onMessageReceived(String topic, String message)
{
    int msg = strtol(message.c_str(),0,10);
    MCP_Write(0x12, state ^= (1u << msg));
}
```

Slika 21. Programski kôd za komunikaciju koristeći MQTT protokol.
Izvor: autor

6.1.2.3 Upravljanje integriranim krugom MCP23017

Za upravljanjem MCP23017 integriranim krugom koristi se I2C komunikacijski protokol. Razvojni okvir Sming, također, ima ugrađenu podršku za I2C komunikacijski protokol koja je ostvarena u biblioteci pod nazivom „Wire“. Prvo se postavljaju svi pinovi kao izlazi te se oni postavljaju na 0. Ovisno u poruci dobivenu od MQTT brokera, postavlja se određeni izlaz, odnosno bit, u registru MCP23017 IC-a. Npr., ako poruka sadrži broj 3, u varijabli „state“ se prebaci

četvrti bit iz trenutnog stanja „0“ u stanje „1“ te se ta varijabla šalje MCP23017 IC-u koji uključi ili isključi relej koji je povezan na izlaz 3.

```
Wire.pins(5, 4);
Wire.begin();
MCP_Write(0x00, 0b00000000);//postavljanje svih nožica kao izlazi
MCP_Write(0x12, 0b00000000); //postavlja sve izlaze na 0
MCP_Write(0x12, state ^= (1u << msg));

void MCP_Write(byte MCPregister, byte MCPdata) {
  Wire.beginTransmission(0x20);
  Wire.write(MCPregister);
  Wire.write(MCPdata);
  Wire.endTransmission();
}
```

Slika 22. Programski kôd za upravljanje MPC23017 IC-om.
Izvor: autor

6.1.3 Trošak komponenti

Tablica 4. Ukupni trošak komponenti za izradu WiFi relej pločice (Cijene na dan 23.08.2016)

Natrijev persulfat	24,70 kn
TES200	14,25 kn
VETRONIT 100 x 160 DVOSTRANA	15,81 kn
USB MINI SMD	5,12 kn
LM3940IS-3.3/NOPB	13,55 kn
MCP23017-E/SS	11,53 kn
ULN2803ADW	4,60 kn
ESP-12F	11,23 kn
SRD-03VDC-SL-C (paket od 10 kom)	34,41 kn
Ukupno:	**135,20 kn**

6.2 Aplikacija za Windows IoT Core

Prva verzija prototipa sadržavala je dvije aplikacije, web poslužitelj koji će posluživati web aplikaciju te aplikacija koje će prikazivati korisničko sučelje pomoću razvojnog sustava Raspberry Pi-u te prikazivati stanje LED diode. Navedene dvije aplikacije se u ovom prototipnom rješenju spajaju i tvore jednu aplikaciju s web sučeljem iz koje se upravlja cijelim sustavom te se time olakšava održavanje te distribucija aplikacije.

Kôd metode „FlipState“ premješta se pod klasu naziva „HttpServer“. Također, koristeći biblioteku naziva „M2Mqtt“, ostvaruje se komunikacija sa MQTT brokerom za komunikaciju s krajnjim uređajima. U metodi naziva „HttpServer“ pokreće se http poslužitelj, definira se metoda koja će se pozvati kod primitka zahtijeva te se ostvaruje konekcija sa MQTT brokerom.

Kako ova verzija prototipnog rješenja sadrži samo jednu aplikaciju, nema trošenja resursa za među-aplikacijsku komunikaciju što bi kod velikog broja zahtjeva moglo usporiti sustav.

```
private const uint BufferSize = 8192;
private int port = 8000;
private readonly StreamSocketListener listener;
private AppServiceConnection appServiceConnection;
private MqttClient client;
private int LEDStatus = 0;

public HttpServer(int serverPort,AppServiceConnection connection)
{
    listener = new StreamSocketListener();
    port = serverPort;
    appServiceConnection = connection;
    listener.ConnectionReceived += (s, e) => ProcessRequestAsync(
e.Socket);
    this.client = new MqttClient("192.168.1.10");
    this.client.Connect(Guid.NewGuid().ToString());
}
…
```

Slika 23. Kod klase „HttpServer“.
Izvor: autor

7. Zaključak

Izgradnjom bežičnog prekidača za sustave pametne kuće nastojalo se pronaći kvalitetno i cjenovno prihvaćeno rješenje za automatiziranje procesa koji se odvijaju o okviru stambenih jedinica s primarnim ciljevima smanjenje ukupne potrošnje energije te stvaranje ugodnije okoline za život i svakodnevne poslove.

Prototipna rješenja koja su izrađena koristeći Windows IoT operacijski sustav instaliran na Raspberry Pi, koji služi kao bazna stanica te sklopovski moduli za bežičnu komunikaciju prema standardu IEEE 802.11 b/g/n, naziva ESP8266 koji služe kao krajnji uređaji, omogućuju da se na vrlo lak i jednostavan način, preko releja upravlja širokim rasponom uređaja. Instalacija sustava se sastoji od spajanja krajnjih uređaja u seriju zajedno sa uređajima se želi upravljati. Krajnji uređaji prototipnih rješenja omogućuju upravljane do 8 različitih visokonaponskih uređaja koristeći pametan telefon ili tablet, standardni prijenosnik ili PC računalo. Komunikacijski protokol MQTT uklapa se savršeno u cjelokupni sustav. On omogućuje brzu, efikasnu i sigurnu komunikaciju kroz cijeli sustav pametne kuće.

Iako je Windows IoT relativno novi operacijski sustav, vidljiv je veliki napredak od prve inačice operacijskog sustava do sada. Također, vide se napori koje tvrtka Microsoft ulaže u Windows IoT operacijski sustav s ciljem širenja programskog sučelja WinRT na sve uređaje, uključujući i ugrađene sustave, čime bi se isplatilo ulaganje u razvoj na Windows IoT platformi.

Izrada sustava za automatizaciju doma je iznimno složen zadatak koji uključuje mnoga znanja iz područja elektrotehnike, elektronike i računarstva pa se može reći da je ovim radom ta tematika samo dotaknuta. Prototipni sustavi izrađeni kao podloga za stvaranje osnovnog koncepta opisanog ovim radom daje temeljne smjernice i s tehničke strane stvara osnovni temelj za razradu sustava.

Kako su izrađeni prototipni sustavi samo uvod u sustave pametnih kuća, ne sadrže mnogo značajki ni različitih modela krajnjih uređaja. Međutim, zahvaljujući komunikacijskom protokolu MQTT, ovakav sustav je proširiv na gotovo neograničen broj uređaja sa kojima se želi upravljati. Također, redovite nadogradnje operacijskog sustava Windows IoT daje nove

značajke sustavu koje se, relativno jednostavno, mogu implementirati u aplikaciju instaliranu na Windows IoT operacijskom sustavu. Daljnji razvoj i nove spoznaje u svim nabrojenim područjima će tehnologije pametne kuće i općenito Interneta stvari učini pristupačnim i široko rasprostranjenim.

8. Popis literature

[1] I. Šumiga, F. Kolarek i D. Srpak, »Inteligentni sustavi za pametnu kuću,« u *Tehnički glasnik*, Varaždin, Sveučilišni centar Varaždin Sveučilište Sjever, pp. 451 - 456.

[2] Ž. Knok, B. Trstenjak i J. Trstenjak, »Inteligentna ili pametna kuća,« u *Zbornik radova Međimurskog veleučilišta u Čakovcu*, Čakovec, Međimursko veleučilište u Čakovcu, pp. 45 - 50.

[3] »Raspberry Pi 2 Model B,« [Mrežno]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Pokušaj pristupa 05 02 2016].

[4] »Alternative Raspberry Pi operating systems,« [Mrežno]. Available: http://thepihut.com/blogs/raspberry-pi-tutorials/17817296-alternative-raspberry-pi-operating-systems. [Pokušaj pristupa 02 02 2016].

[5] »ESP8266 module family,« [Mrežno]. Available: http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family. [Pokušaj pristupa 06 02 2016].

[6] »ESP8266EX Datasheet Ver. 4.3,« [Mrežno]. Available: http://www.esp8266.com/wiki/lib/exe/fetch.php?media=0a-esp8266_datasheet_en_v4.3.pdf. [Pokušaj pristupa 05 03 2016].

[7] »ATmega48A/PA/88A/PA/168A/PA/328/P.,« [Mrežno]. Available: http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Summary.pdf. [Pokušaj pristupa 05 03 2016].

[8] »NodeMCU,« [Mrežno]. Available: https://github.com/nodemcu/nodemcu-firmware. [Pokušaj pristupa 16 09 2016].

[9] Koblan, Kolban's Book on ESP8266, 2016.

[10] »Sming,« [Mrežno]. Available: https://github.com/SmingHub/Sming. [Pokušaj pristupa 22 8 2016].

[11] »Windows 10 for the Internet of Your Things,« [Mrežno]. Available: https://www.microsoft.com/en-us/WindowsForBusiness/windows-iot. [Pokušaj pristupa 06 03 2016].

[12] »How-to guides for UWP apps on Windows 10,« [Mrežno]. Available: https://msdn.microsoft.com/en-us/windows/uwp/index. [Pokušaj pristupa 06 03 2016].

[13] »MQTT Essentials Part 2: Publish & Subscribe,« [Mrežno]. Available: http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe. . [Pokušaj pristupa 08 02 2016].

[14] »Frequently Asked Questions,« [Mrežno]. Available: http://mqtt.org/faq. [Pokušaj pristupa 06 03 2016].

[15] »Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry,« IBM, 9 2012. [Mrežno]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf. [Pokušaj pristupa 22 8 2016].

[16] »OASIS,« 29 10 2014. [Mrežno]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. [Pokušaj pristupa 22 8 2016].

[17] »SRD-03VDC-SL-C Datasheet,« [Mrežno]. Available: http://www.datasheetspdf.com/PDF/SRD-03VDC-SL-C/720555/1. [Pokušaj pristupa 25 8 2016].

[18] V. Pinter, u *Osnove elektrotehnike*, Zagreb, ITP "Tehnička knjiga" D.D, 1994, pp. 248-250.

[19] »M2Mqtt,« [Mrežno]. Available: https://m2mqtt.wordpress.com/. [Pokušaj pristupa 16 09 2016].

Prilog 1 – ESP-12F kôd (Arduino)

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define wifi_ssid "demo"
#define wifi_password "ESP-8266"

#define mqtt_server "192.168.1.60"
#define mqtt_user ""
#define mqtt_password ""

#define topic_pub "mqtt/test1"

WiFiClient espClient;
PubSubClient client(espClient);
int LED = 14;
int RELAY = 16;
int msgCount = 0;
bool st = false;
void setup_wifi();
void callback(char* topic, byte* payload, unsigned int length);
void setColor(char c);

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(RELAY, OUTPUT);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

}

void setup_wifi() {
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);

  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(wifi_ssid);

  WiFi.begin(wifi_ssid, wifi_password);

  while (WiFi.status() != WL_CONNECTED) {
    setColor('B');
    delay(500);
    Serial.print(".");
```

```
  }

  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    setColor('O');
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      setColor('G');
      client.subscribe("mqtt/test2");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setColor(char c)
{
  st = false;
  if(c == 'G' || c == 'g')  {
    Serial.write("\nZeleno\n");
    digitalWrite(LED, HIGH);
  } else if(c == 'O' || c == 'o') {
    Serial.write("\nOrange\n");
    digitalWrite(LED, LOW);
  } else if((c == 'B' || c == 'b'))  {
    Serial.write("\nBlink\n");
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
  }
}

void FlipLED(bool state)
{
  if (state)   {
    Serial.println("Turning LED ON");
    digitalWrite(RELAY, HIGH);
    Serial.println("LED ON");
  }
  else if(!state)  {
```

```
    Serial.println("Turning LED OFF");
    digitalWrite(RELAY, LOW);
    Serial.println("LED off");
  }
  else return;

}

void callback(char* topic, byte* payload, unsigned int length)
{
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++)  {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  if ((char)payload[0] == '1')  {
    FlipLED(1);
  } else  {
    FlipLED(0);
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

}
```

Prilog 2 – ESP-12F kôd (Sming)

```
#include <user_config.h>
#include <SmingCore/SmingCore.h>
#include <string>
#include <cstdlib>

#define WIFI_SSID "Code"
#define WIFI_PWD "thebigbangtheory"

void startMqttClient();
void onMessageReceived(String topic, String message);
void MCP_Write(byte MCPregister, byte MCPdata);

int state = 0b00000000;

Timer procTimer;

MqttClient mqtt("192.168.1.11", 1883, onMessageReceived);

inline bool isInteger(const String & s)
{
   if(s == NULL || ((!isdigit(s[0])) && (s[0] != '-') && (s[0] !=
'+'))) return false ;

   char * p ;
   strtol(s.c_str(), &p, 10) ;

   return (*p == 0) ;
}

void onMessageReceived(String topic, String message)
{
      Serial.print(topic);
      Serial.print(":\r\n\t");
      Serial.println(message);

      if(!isInteger(message))
      {
            Serial.print("\n ERROR: Message not recognized!");
            return;
      }
      int msg = strtol(message.c_str(),0,10);
      if(msg < 0 || msg > 7) return;

      MCP_Write(0x12, state ^= (1u << msg));
}

void startMqttClient()
```

```
{
      if(!mqtt.setWill("last/will","The connection lost!", 1,
true)) {
            debugf("Unable to set the last will and testament.");
      }
      mqtt.connect("esp8266");
      mqtt.subscribe("wifi/relayBoard/#");
}

void MCP_Write(byte MCPregister, byte MCPdata) {

  Wire.beginTransmission(0x20);
  Wire.write(MCPregister);
  Wire.write(MCPdata);
  Wire.endTransmission();
}

void connected()
{
      Serial.println("CONNECTED");
      startMqttClient();
}

void notConnected()
{
      Serial.println("FAILED TO CONNECT!");
}

void init()
{
      Serial.begin(SERIAL_BAUD_RATE);
      Serial.systemDebugOutput(true);

      WifiStation.config(WIFI_SSID, WIFI_PWD);
      WifiStation.enable(true);
      WifiAccessPoint.enable(false);

      Wire.pins(5, 4);
      Wire.begin();

      MCP_Write(0x00, 0b00000000);
      MCP_Write(0x12, 0b00000000);

      WifiStation.waitConnection(connected, 20, notConnected);
}
```

Prilog 3 – Kôd aplikacije instalirane na Windows IoT OS-u

```
using System;
using System.Text;
using Windows.Foundation.Collections;
using Windows.ApplicationModel.Background;
using Windows.ApplicationModel.AppService;
using Windows.Networking.Sockets;
using System.IO;
using Windows.Storage.Streams;
using System.Threading.Tasks;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using uPLibrary.Networking.M2Mqtt;
namespace WebServerTask {
  public sealed class WebServerBGTask: IBackgroundTask {
    public void Run(IBackgroundTaskInstance taskInstance) {
      taskInstance.Canceled += OnCanceled;
      serviceDeferral = taskInstance.GetDeferral();
      MqttBroker broker = new MqttBroker();
      broker.Start();
      var    appService    =    taskInstance.TriggerDetails    as
AppServiceTriggerDetails;
      if (appService != null && appService.Name ==
        "App2AppComService") {
        appServiceConnection = appService.AppServiceConnection;
        appServiceConnection.RequestReceived                    +=
OnRequestReceived;
      }
    }
    private  async  void  OnRequestReceived(AppServiceConnection
sender,
      AppServiceRequestReceivedEventArgs args) {
      var message = args.Request.Message;
      string command = message["Command"] as string;
      switch (command) {
        case "Initialize":
          {
            var messageDeferral = args.GetDeferral();
            var returnMessage = new ValueSet();
            server = new HttpServer(8000, appServiceConnection);
            IAsyncAction              asyncAction              =
Windows.System.Threading.ThreadPool
            .RunAsync(
              (workItem) => {
                server.StartServer();
              });
            returnMessage.Add("Status", "Success");
```

```
            var          responseStatus          =          await
args.Request.SendResponseAsync(
              returnMessage);
            messageDeferral.Complete();
            break;
          }
        case "Quit":
          {
            serviceDeferral.Complete();
            break;
          }
      }
    }
    private void OnCanceled(IBackgroundTaskInstance sender,
      BackgroundTaskCancellationReason reason) {}
    HttpServer server;
    BackgroundTaskDeferral serviceDeferral;
    AppServiceConnection appServiceConnection;
  }
  public sealed class HttpServer: IDisposable {
    string HtmlString =
      "<html><head><title>Blinky    App</title></head><body><form
action=\"blinky.html\"   method=\"GET\"><input   type=\"checkbox\"
name=\"state\"                                        value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  1<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  2<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  3<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  4<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  5<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  6<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz  7<br><input  type=\"checkbox\"  name=\"state\"  value=\"0\"
onclick=\"((this.checked)?this.setAttribute('value',
'0'):this.setAttribute('value',    '1'));    this.form.submit()\">
Izlaz 8<br></form></body></html>";
    private
```

```
    const uint BufferSize = 8192;
    private int port = 8000;
    private readonly StreamSocketListener listener;
    private AppServiceConnection appServiceConnection;
    private MqttClient client;
    private int LEDStatus = 0;
    public   HttpServer(int   serverPort,   AppServiceConnection
connection) {
      listener = new StreamSocketListener();
      port = serverPort;
      appServiceConnection = connection;
      listener.ConnectionReceived += (s, e) =>
        ProcessRequestAsync(e.Socket);
      this.client = new MqttClient("192.168.1.10");
      this.client.Connect(Guid.NewGuid().ToString());
    }
    public void StartServer() {#
      pragma warning disable CS4014
      listener.BindServiceNameAsync(port.ToString());#
      pragma warning restore CS4014
    }
    public void Dispose() {
      listener.Dispose();
    }
    private async void ProcessRequestAsync(StreamSocket socket) {
      StringBuilder request = new StringBuilder();
      using(IInputStream input = socket.InputStream) {
        byte[] data = new byte[BufferSize];
        IBuffer buffer = data.AsBuffer();
        uint dataRead = BufferSize;
        while (dataRead == BufferSize) {
          await input.ReadAsync(buffer, BufferSize,
            InputStreamOptions.Partial);
          request.Append(Encoding.UTF8.GetString(data,         0,
data.Length));
          dataRead = buffer.Length;
        }
      }
      using(IOutputStream output = socket.OutputStream) {
        string requestMethod = request.ToString().Split('\n')[0];
        string[] requestParts = requestMethod.Split(' ');
        if (requestParts[0] == "GET")
          await WriteResponseAsync(requestParts[1], output);
        else
          throw new InvalidDataException(
            "HTTP method not supported: " +
            requestParts[0]);
      }
    }
```

```
    private async Task WriteResponseAsync(string request,
      IOutputStream os) {
      string state = "Unspecified";
      bool stateChanged = false;
      if (request.Contains("blinky.html?state=on")) {
        state = "On";
        stateChanged = true;
      } else if (request.Contains("blinky.html?state=off")) {
        state = "Off";
        stateChanged = true;
      }
      if (stateChanged) {
        var updateMessage = new ValueSet();
        updateMessage.Add("State", state);
        var           responseStatus           =           await
appServiceConnection.SendMessageAsync(
          updateMessage);
      }
      string html = state == "On" ? onHtmlString : offHtmlString;
      using(Stream resp = os.AsStreamForWrite()) {
        byte[] bodyArray = Encoding.UTF8.GetBytes(html);
        MemoryStream stream = new MemoryStream(bodyArray);
        string header = String.Format("HTTP/1.1 200 OK\r\n" +
          "Content-Length: {0}\r\n" +
          "Connection: close\r\n\r\n",
          stream.Length);
        byte[] headerArray = Encoding.UTF8.GetBytes(header);
        await           resp.WriteAsync(headerArray,           0,
headerArray.Length);
        await stream.CopyToAsync(resp);
        await resp.FlushAsync();
      }
    }
    private void FlipLED() {
      if (LEDStatus == 0) {
        LEDStatus = 1;
        this.client.Publish("mqtt/test2", Encoding.UTF8.GetBytes(
          "1"));
      } else {
        LEDStatus = 0;
        this.client.Publish("mqtt/test2", Encoding.UTF8.GetBytes(
          "0"));
      }
    }
  }
}
```